Süsteemprogrammeerimine keeles C

Loeng 15

Milles räägitakse hästi palju asju hästi pealiskaudsel viisil üle, et tekiksid seosed ja mõningane tülpimus

Tänases osas

- C keel
- C librad

C keel

Ehk see osa, mis ei ole librad

Kompileerimine

- C programmi kompileerimine koosneb neljast etapist
  - *preprocessing* (# -ga algavad read)
  - *compilation proper* (tulemuseks programm assembleris)
  - *assembly* (tulemuseks objektkood)
  - *linking* (tulemuseks midagi käivitatavat)

Andmetüübid

- Primitiivandmetüübid
  - int
  - float
  - double
  - char
- Lisaparameetrid
  - long
  - short

Andmetüübid

- struct
  - elementidele viitamine (. ja ->)
  - Bitiväljad (:1)
  - (mäluhõive)
- Union
  - (mäluhõive)
- Lisaks
  - void
  - enum

Pointerid

- Pointer on muutuja, milles on mäluaadress mingit tüüpi andmetele
- Saab kasutada kahel viisil:
  - mäluaadressi salvestamisel ja lugemisel
  - salvestatud mäluaadressi poolt viidatud mälu salvestamisel ja lugemisel
- & ja * operaatorid
- Pointeraritmeetika
- Pointer funktsioonile

Massiivid

- Deklareerimine

```
int a[10];
int b[] = { 1, 2, 3, 4, 5};
int c[][] = { {1,2}, {2,3}, {4, 5}};
```

- Paiknevad mälus järjest
- Mitmemõõtmelised (adresseerimine):

```
tabel[rida][tulp];
*(*(tabel + rida) + tulp );
```

- Massiivi tüüp on pointer vastavat tüüpi andmetele
- Massiivi elementidele võib viidata nii pointeri kui ka massiivi süntaksit kasutades

Stringid

- C keeles on string kokkuleppeline peaaegu keeleväline konstruktsioon.
- String on char'ide massiiv, mille lõppu tähistab '\0'
- Initsialiseerimise mugavamaks tegemiseks on meil {'s', 'e', 'l', 'l', 'i', 's', 'e', '\0'} asemel võimalik „selline“ kirjutada.

Mälu klassid (*Storage class*)

- Määravad muutujate skoobi (nähtavuse) ja paiknemise mälus
- auto – vaikimisi klass lokaalsetele muutujatele, hävib pärast deklareerivast blokist väljumist, *stack*
- register – protsessoriregistris hoitav muutuja
- static – vaikimisi klass globaalsete muutujatele, initsialiseeritakse käivitumisel, funktsioonides kasutatuna säilitavad väärtuse väljakutsete vahel
- extern – ütleb, et muutuja jaoks eraldati mälu mingis teises moodulis, mis ilmub linkimisel

const tüübid

- const tähistab, et pärast initsialiseerumist antud muutujat ei muudeta
  - millal on parem kui enum või makro?
    - väärtus selgub programmi jooksmise käigus
    - kasutatakse kohas, kus on vaja & aadressioperaatorit
    - kompilaator/debugger peab märkama
    - püüdes sundida funktsiooni mitte muutma oma argumente, näiteks massiivi elemente
- NB! const char *s ütleb, et s poolt viidatud baite ei tohi muuta. Pointerit ennast võime me muuta! Kui tahame muutumatut pointerit, deklareerime char* const s.

typedef

- deklareerib mingi tüübi jaoks uue tüübinime
- hõlbustab hilisemaid muutmisi ja struct puhul parandab veidi loetavust
- typedef struct tree Tree;

void

- void funktsiooni tagastustüübina tähistab, et funktsioon ei tagasta midagi
- void * abil märgitakse pointerit, mille tüüp selgub programmi käivitamise käigus (*cast*i abil kasutatakse).
- void pointerit saab teisendada pointeriks suvalisele teisele andmetüübile

Expression, Statement

- Expression = operand, operator. Tehe Jaguneb:
  - Võrdlustehted: x>y, 2==y, x!=5
  - Aritmeetilised tehted: x+2, y--, j*j, 6/3
  - Omistamistehe: x=y, x=4
  - **NB!** Ära võrdle nii: x=y /* VALE! */
- Statement – programmilause
  - lihtne x=y;
  - liit- {x=5; y=z=3; f* = 5.0; }
  - tsüklid for(..) , while(..)
  - if-laused if, if..else

Tehted

```
>>    () [] -> .
<<    ! ~ ++ -- + - * & (type) sizeof
>>    * / %
>>    + -
>>    << >>
>>    < <= > >=
>>    == !=
>>    &
>>    ^
>>    |
>>    &&
>>    ||
<<    ? :
<<    = += -= *= /= %= &= ^= |= <<= >>=
>>    ,
```

Tüüpide teisendamisest

- Automaatne teisendamine: kui on eritüübilised operaatorid
- Üldreegel teisendamisel: operaatorid teisendatakse kõige "võimekamaks"
- long double > double > float > long > int > short
- short ja char teisenduvad int-iks
- *signed* ja *unsigned*: tulemus märgita arv, negatiivsed arvud lähevad väga suureks
- Vägisi teisendamiseks on *cast*imine:

```
f = i / (float)3 ;
```

Bittoperaatorid

Opereerivad täisarvudel bitikaupa
Kasutage unsigned täisarve, väldite sellega raskestileitavaid vigu
| - Bitwise OR
& - Bitwise AND
^ - Bitwise XOR
~ - Bitwise complement (igal bitil NOT)
<< - Bitwise shift left
>> - Bitwise shift right
Nihked on kahega korrutamised ja jagamised

Bittide testimine

Bitti testimiseks tehakse bititehe mingi maskiga

Biti ülaloleku testiks kasutame & tehet maskiga

Maske saab luua #define käsuga

Biti üleslükkamiseks omistatakse mask | tehtega
Mahavõtmine on & tehe ja ~(mask)
Lülitamine on ^ mask

Deklareerimine

- Enne muutuja ja funktsiooni kasutamist tuleb nad deklareerida, et kompilaator teaks ette valmistuda
  - Deklareerimine ütleb, mis andmetüüpe kasutatakse
  - Deklareerimata funktsiooni kasutamisel määratakse selle andmete tüübiks ja tagastuvaks tüübiks *int*.

```
int x;
struct tree top_node;

void stop_and_catch_fire(long when);
```

Funktsioonide argumendid

- Funktsiooni väljakutsel kopeeritakse funktsioonile antud argumendid *stack*i automaatseteks muutujateks. Edastatakse väärtuse *koopia*. (*call-by-value)*
- Siit lähtub, et funktsioonist väljudes kaob nende väärtus, sest skoop, milles nad eksisteerivad, hävib
- Kui funktsioon peab oma argumente muutma, antakse pointer muudetavatele andmetele (*call-by-reference*)
- Massiiv on tüübilt pointer, mis sellest järeldub?

Funktsiooni main() argumendid

- Funktsioon main() saab "välismaailmast" parameetreid. Parameetrite tüüp on ettemääratud:

```
main (int argc, char ** argv) {…}
```

  - argc – argumentstringide arv
  - argv – argumentstringide massiiv, argv[0] on alati programminimi
- Programmi käivitamine "*minuproge arg1 arg2*"
  - argc = 3
  - argv[0] = "minuproge"
  - argv[1] = "arg1"
  - argv[2] = "arg2"

Skoop

Bloki skoop { ... }
Blokis ja selle alamblokkides

Funktsiooni skoop
Funktsiooni ulatuses (sisuliselt sama mis eelmine)

Faili skoop
Antud faili ulatuses

Prototüübi skoop
Ainult prototüübi ulatuses

Preprotsessor

- Töötab enne kompileerimist
- #define KONSTANT 2
- #define MAKRO(a) ((a)+(a))
  - asendub programmikoodi täpselt: võib põhjustada mitmekordset käivitumist, programmi mittekompileerumist ja tehete järjekorra probleeme
- #ifndef, #ifdef, #endif – tingimuslik lisamine
- #if, #else, #elif, #endif
- #undef – tühistab definitsiooni

C librad

Ehk see, mis on kah vaja eksamiks teada

Stream

- <stdio.h>
- stdin, stdout, stderr – standardsed streamid
- Streami tüübiks on FILE
- ehitatud failideskriptorite peale – puhvritega ja mugavad
- fopen(), fclose()
- printf(), scanf(), getchar(), putchar(), gets(), puts()
- fprintf(), fscanf(), getc(), putc(), fgets(), fputs()
- fread(), fwrite(), tmpfile(), fflush()
- perror()

<ctype.h>

- `int isalnum(int c);`
- `int isalpha(int c);`
- `int iscntrl(int c);` is control character. In ASCII, control characters are `0x00` (`NUL`) to `0x1F` (`US`), and `0x7F` (`DEL`)
- `int isdigit(int c);` is decimal digit `int`
- `isgraph(int c);` is printing character other than space
- `int islower(int c);` is lower-case letter
- `int isprint(int c);` is printing character (including space). In ASCII, printing characters are `0x20` (`' '`) to `0x7E` (`'~'`)
- `int ispunct(int c);` is printing character other than space, letter, digit
- `int isspace(int c);` is space, formfeed, newline, carriage return, tab, vertical tab
- `etc.`

<math.h>

- `double exp(double x);` *x* aste
- `double log(double x);` *x* naturaallogaritm
- `double log10(double x);` *x* kümnendlogarigm
- `double pow(double x, double y);` *x* astmes *y*
- `double sqrt(double x);` *x* ruutjuur
- `double ceil(double x);` *x* ümardatuna üles
- `double floor(double x);` *x* ümardatuna alla
- `double fabs(double x);` *x* absoluutväärtus
- `double ldexp(double x, int n);` *x* korda 2 astmel *n*
- `double sin(double x);`
- `double cos(double x);`
- `double tan(double x);`
- `double sinh(double x);`
- `double cosh(double x);`
- `double tanh(double x);`
- *jne*

<stdlib.h>

- malloc()
- realloc()
- calloc()
- free()
- võetud mälu (kui calloc() välja jätta) on initsialiseerimata

<string.h>

- erinevad stringifunktsioonid
- eeldavad, et \0 on lõpus
- strlen(), strcpy(), strncpy(), strcmp() (NB! mida teeb), strncmp()
- strtok()
- strcat(), strncat()
- strstr(), strpbrk()
- teisi on veel, aga nendega ei piina

Madalatasemeline I/O

- failideskriptorid
- 0, 1, 2 vastavad stdin, stdout, stderr streamidele
- open(), creat(), close()
- fdopen(), fileno()
- read(), write()
- lseek()
- dup(), dup2()

Süsteemikäsud <unistd.h>

- link(), symlink(), unlink()
- stat(), fstat(), lstat(), access()
- chmod(), fchmod()
- chown(), fchown()
- umask()
- mkdir(), rmdir(), chdir(), fchdir(), getcwd()

<time.h>

- time_t, clock_t, struct tm andmetüübid
- clock(), time(), gmtime(), localtime()
- ctime(), asctime(), difftime()
- mktime()

Võrgundus

- socket()
- PF_LOCAL, PF_INET
- SOCK_RAW, SOCK_DGRAM, SOCK_STREAM
- send(), recv(), sendto(), recvfrom()
- connect(), shutdown()
- htons(), htonl(), ntohs(), ntohl()
- gethostbyname(), gethostbyaddr()
- bind(), listen(), accept()
- socketpair()

<stdarg.h>

- Varieeruva argumentide arvuga funktsioonide puhul kasutatavad makrod (ja kuidas töötavad)

```
void va_start(va_list ap, last);
type va_arg(va_list ap, type);
void *va_end(va_list ap);
```

Protsessid

- exec() perekond
- fork()
- wait()
- atexit(), exit()
- getuid(), geteuid(), getgid(), getegid(), getpid(), getppid()

Signaalid

- signal()
- kill()
- alarm()
- Handlerid
  - Lõpetav
  - Hüppega
  - Tööd jätkav

Long jump

- <setjmp.h>
- jmp_buf_t
- setjmp()
- longjmp()

Lõimed

- <pthread.h> POSIX – threads
- pthread_create(), pthread_self()
- pthread_exit(), pthread_cancel(), pthread_detach(), pthread_join()
- pthread_mutex_init(), pthread_mutex_lock(), pthread_mutex_unlock()
- pthread_cond_init(), pthread_cond_wait(), pthread_cond_signal(), pthread_cond_broadcast()
- pthread_atfork()
- *lõimepõhiseid muutujaid ei küsi (pthread_key_ seeria)*

Getopt

- getopt() kasutamine
- getopt_long() - seda ei küsi

Make

- Make on utiliit kompileerimiseks
- Tehakse programmi kompileerimise sõltuvisi kirjeldav makefile (või Makefile) nimeline fail

```
target: prerequisites ...
        command
        ...
```

stdlib.h (cont)

```
void* bsearch(const void* key , const void*
  base ,size_t n,size_t size , int (* cmp )
  (const void*, const void*));
```

- Searches ordered array *base* (of *n* objects each of size *size*) for item matching *key* according to comparison function *cmp*. *cmp* must return negative value if first argument is less than second, zero if equal and positive if greater. Items of *base* are assumed to be in ascending order (according to *cmp*). Returns a pointer to an item matching *key* , or `NULL` if none found.

stdlib.h (cont)

```
void qsort(void* base ,size_t n,size_t size ,
  int (* cmp )(const void*, const void*));
```

- Arranges array *base* (of *n* objects each of size *size*) into ascending order according to comparison function *cmp*. *cmp* must return negative value if first argument is less than second, zero if equal and positive if greater.

qsort example

```
#include < stdlib.h >
#include < stdio.h >
long a[1000];

int compfunc(const void *x, const void *y){
   if (a[*(int *)x] < a[*(int *)y]) return -1;
   else if (a[*(int *)x] == a[*(int *)y]) return 0;
   else return 1; }

void main() {
   int b[1000]; int i;
   for (i=0; i<1000 ; i++) {
      a[i]=(long)rand(); b[i] = i; }
   qsort(b, 1000, sizeof(int), compfunc);
   for (i=0 ; i < 1000 ; i++)   {
      printf("%ld ",a[b[i]]);
       }
}
```

Läbi

Teksti lisamiseks klõpsi siin